

Specific proposals for the use of Petri Nets in a Concurrent Programming Course

João Paulo Barros

`jpb@estig.ipbeja.pt`

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Beja

PORTUGAL



Objective

Improve student understanding of
concurrency concepts

How?

Using (also) Petri nets

Students like it!

This Presentation

1. FSP (a process algebra)
2. Petri nets
3. Course Structure
4. The proposals
 - Synchronization and Deadlock
 - Composition
 - Conflict and Starvation
5. Evaluation
6. Conclusions and Future Work

FSP (Finite State Processes)

- It's a process algebra

"(...)specifically designed to facilitate modelling of finite state processes as Labeled Transition Systems."

in "Jeff Magee and Jeff Kramer, "Concurrency State Models and Java Programs", John Wiley & Sons, 1999.

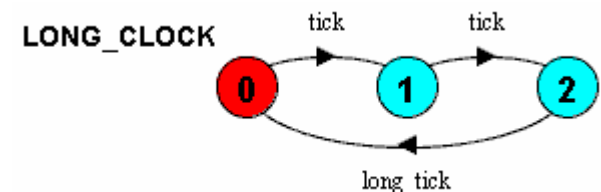
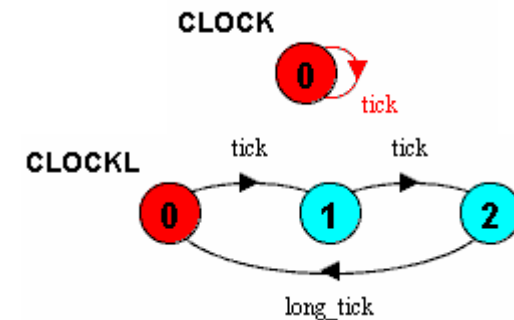
Example:

CLOCK = (tick -> CLOCK).

CLOCKL(N = 2) = LG[0],

LG[lg:0..N] = (**when** (lg < N) tick -> LG[lg+1]
| **when** (lg== N) long_tick -> CLOCKL).

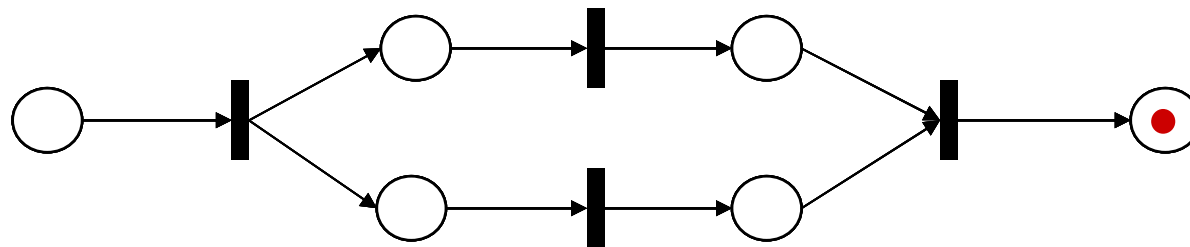
||LONG_CLOCK = (CLOCK || CLOCKL).



Petri nets

- Can be seen as generalized state machines
 - Transitions are nodes and can have more than one input arc and/or output arc
 - Model parallelism and synchronization explicitly!

Example:



Course Structure 1/2

Based on the book:

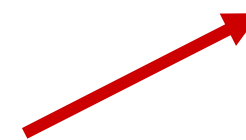
Jeff Magee and Jeff Kramer,
“**Concurrency State Models and
Java Programs**”,
John Wiley & Sons, 1999.

Course Structure 2/2

1. Concepts (e.g. synchronization, deadlock)
2. Relevant FSP constructs
- 3. Relevant Petri net constructs**
4. FSP model and analysis using the LTSA tool
- 5. Construction of the Petri net model from the FSP specification and/or from the state machines generated by the LTSA tool.**
- 6. Simulation of the Petri net model**
7. Translation to a UML class or object diagram
8. Coding of the Java program

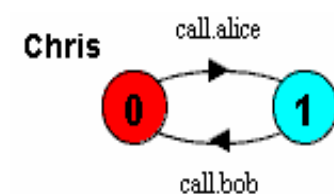
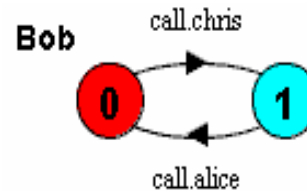
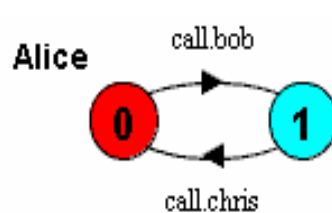
Synchronization and Deadlock 1/2

```
Alice = (call.bob  -> wait.chris -> Alice).  
Bob   = (call.chris -> wait.alice -> Bob).  
Chris = (call.alice -> wait.bob   -> Chris).  
  
||s = (Alice || Bob || Chris) /{call/wait}.
```

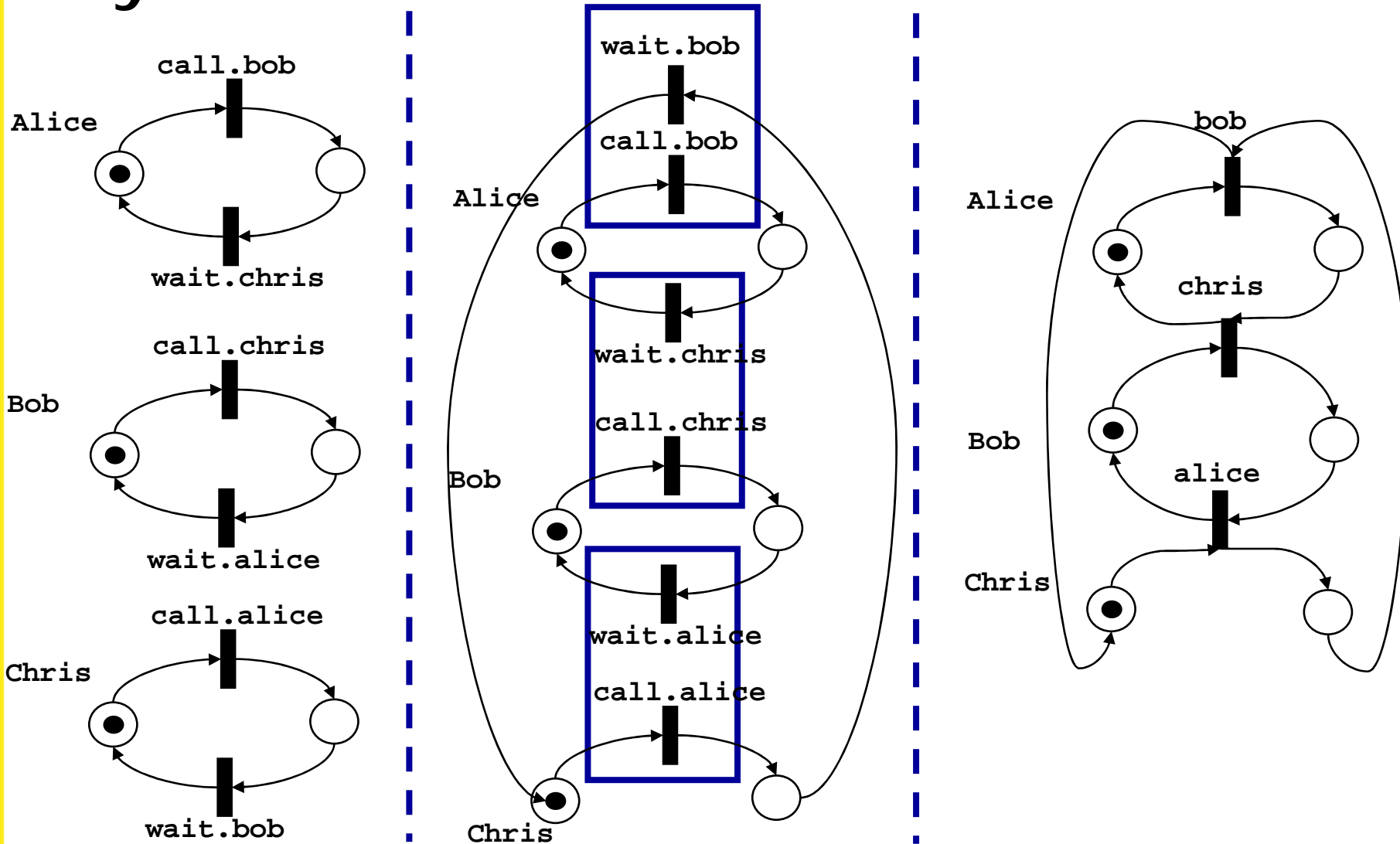


Deadlock?

Where?
Why?



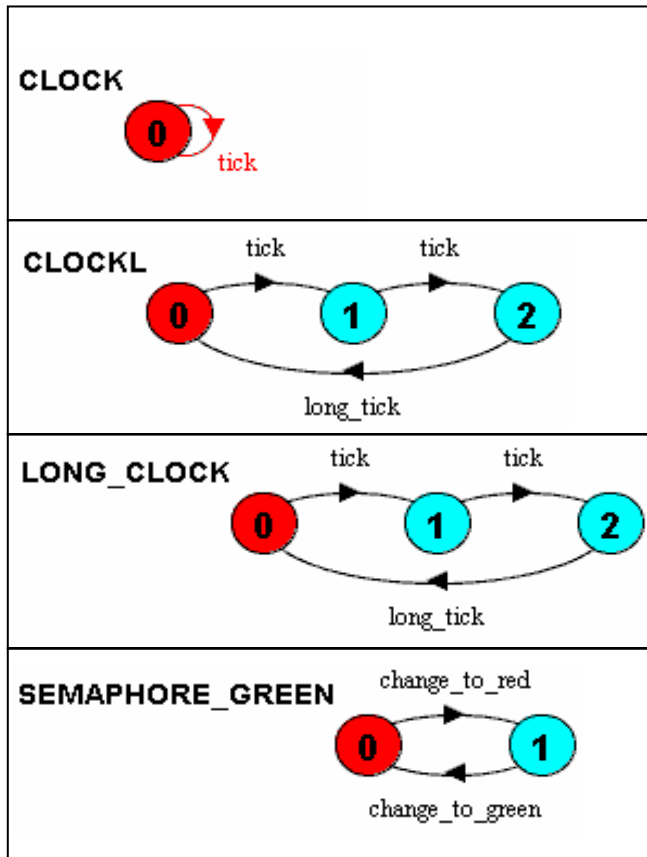
Synchronization and Deadlock 2/2



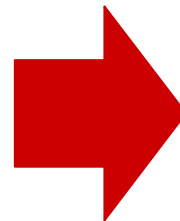
Composition 1/2

Wait for a **tick** in GREEN state

Wait for a **long_tick** (two **ticks**) in RED state

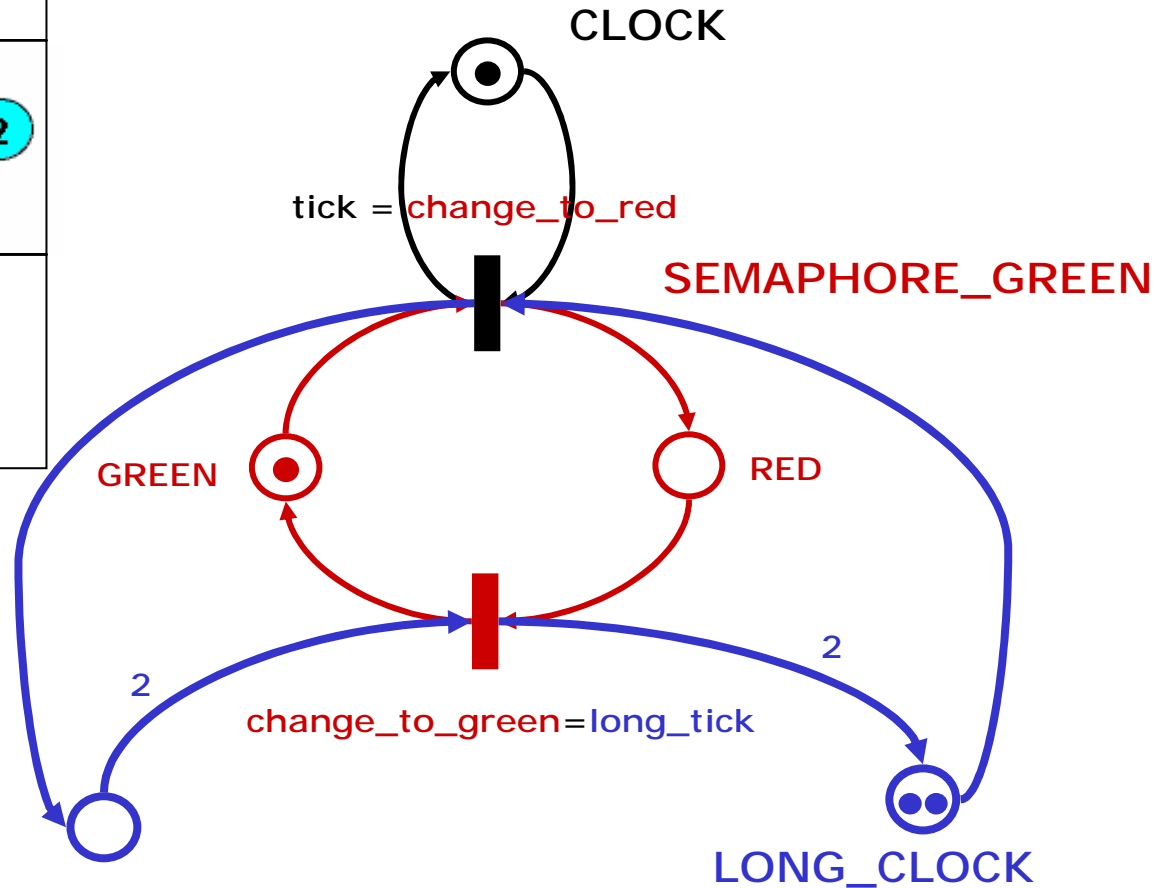
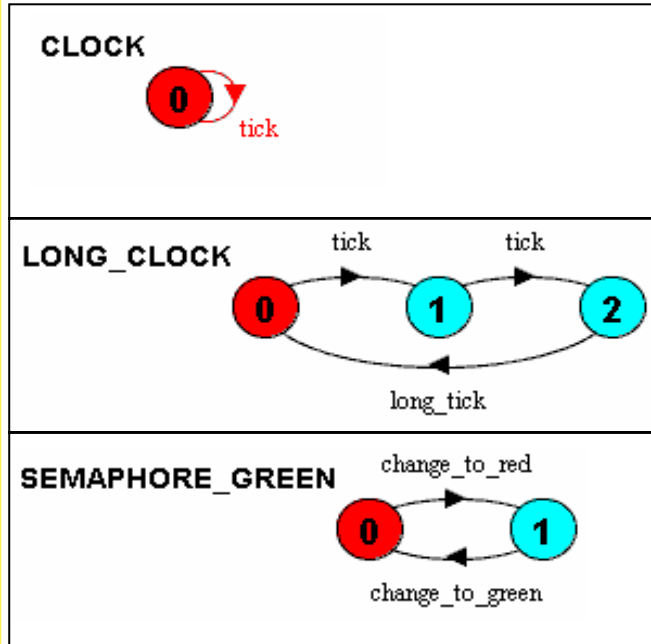


|| SEMAPHORE_TIMED =
(CLOCK || LONG_CLOCK || SEMAPHORE_GREEN)
/{ change_to_red/tick,
change_to_green/long_tick }.



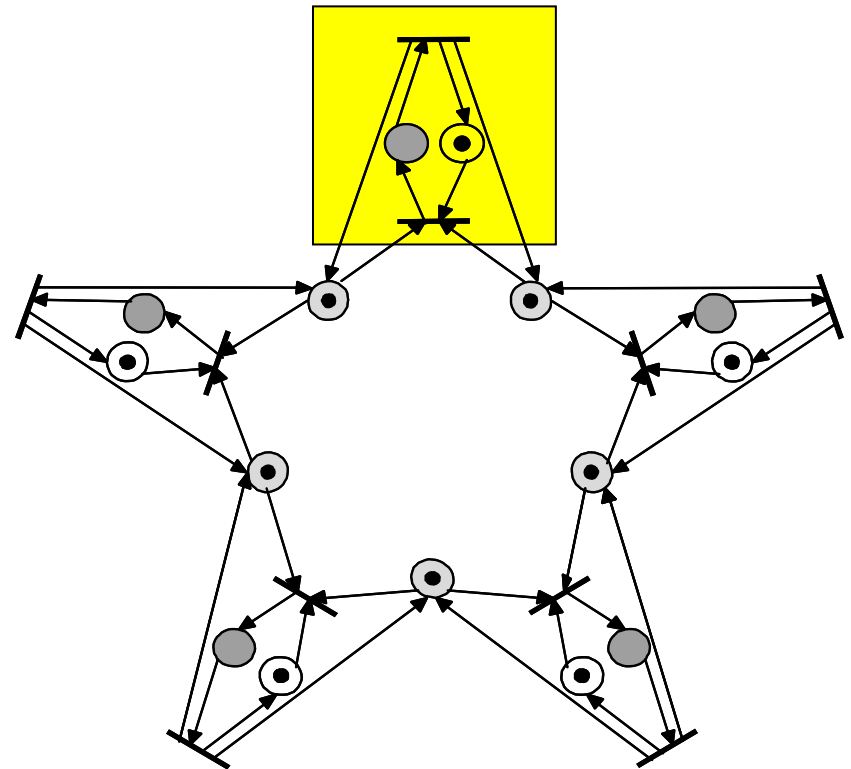
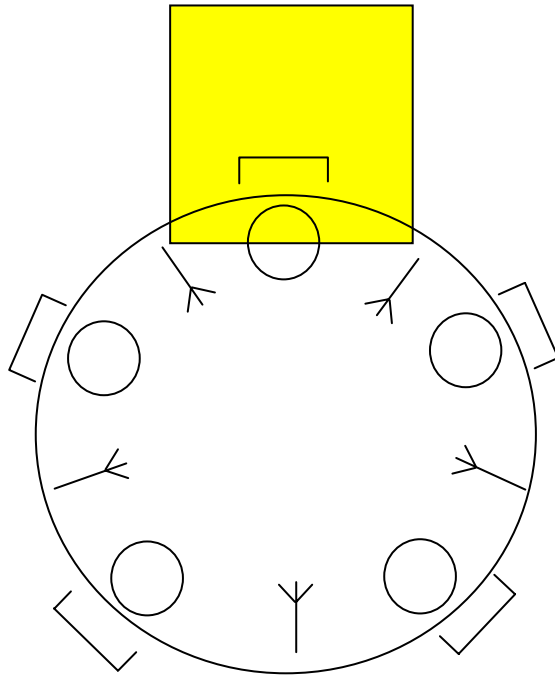
Why?

Composition 2/2



Conflict and Starvation

The dining philosophers!



Evaluation

Students inquiry conclusions :

☺ The addition of Petri nets was considered useful in the understanding of concepts

☹ Petri nets models are not easier to build than FSP models

☺ Easier to start by the Petri net model (as opposed to the FSP model)

☺ Petri net models make the final Java programs easier to understand

☹ *Only 10 replies in 16 possible...*

Conclusions

- Petri nets improved the understanding of process composition, synchronization, deadlock, conflict and starvation .
- Use of a Petri net graphical editor with a “token player” is extremely recommended.

Future Work

- A tool for translating FSP specifications to Petri nets.
- Development of Petri net models for the FSP examples in the book.